國立政治大學商學院資訊管理學系

碩士學位論文

以類神經網路解決情境式推薦問題

A Neural Network Approach to the Contextual-Bandit Problem

指導教授: 林怡伶博士 蕭舜文博士

研究生：陳高欽 撰

中 華 民 國 109 年 7 月

DOI:10.6814/NCCU202001541

Acknowledgement

To my family,

To my advisor, Dr. Yi-Ling Lin & Dr. Shun-Wen Hsiao, giving me guidance and helping me to complete my research,

To the thesis committee, Dr. Chien-Chin Chen,

To the community that is exploring contextual bandit and neural network.

DOI:10.6814/NCCU202001541

摘要

為了向用戶提供合適的商品推薦，推薦系統已在市場中廣泛應用。儘管市場上衝刺著各種可以使用的數據分析，但是冷啟動（Cold Start）問題對於新進用戶來說仍然是一個大問題。許多最新的推薦算法在假設用戶和商品保持線性關係的前提下設計了演算法，而實際上大多數情況下兩者間存在非線性關係。這項研究開發了一種使用神經網絡（NN）和情境式是推薦的演算法來處理非線性特徵和探索利用的權衡。推薦系統可以有效地預測新進用戶的喜好，還可以快速探索快速變化的喜好。通過將貝葉斯網絡（Bayesian networks）和自動編碼器（AE）集成到NN中，我們的系統, NN Contextual Bandit（NNCB）可以利用不同程度的探索和開發。因此，我們的系統能快速適應情境的變化。我們採用真實世界中的影片評分數據集來證明所提出系統的有效性，與傳統的情境是推薦演算法相比，該系統大約4%的優於就演算法。

關鍵字: 情境式推薦, 多選項推薦, 神經網路, 推薦系統

DOI:10.6814/NCCU202001541

ABSTRACT

Recommendations have been wildly applied in marketplaces to provide right items to users. While various heterogeneous data available in marketplaces, the cold start problem is still a big issue for newcomers. Many state-of-the-art recommendation algorithms were designed on the assumption that users and items remain a linear relationship, while most cases exist nonlinear relationship in reality. This study develops an algorithm using neural network (NN) and contextual bandit to deal with nonlinear context and explore-exploit tradeoff. The recommendation system could effectively predict newcomers' preferences and also provide quick exploration for fast-changing preferences. By integrating Bayesian networks and AutoEncoder (AE) in the NN, our system, NN Contextual Bandit (NNCB), could leverage different levels of exploration and exploitation. Thus, the proposed recommendation can quickly adapt to the real-time context. We adopt real-world video rating dataset to demonstrate the effectiveness of the proposed system which improve 4% regret as the conventional bandit algorithms.

Keyword: contextual bandit, multi-armed bandit, neural network, recommendation system

DOI:10.6814/NCCU202001541

Table of Contents

DOI:10.6814/NCCU202001541

List of Tables

DOI:10.6814/NCCU202001541

List of Figures

DOI:10.6814/NCCU202001541

CHAPTER 1 INTRODUCTION

With the development of new technologies, handling big data generated by customers has become one of the most important issues for 91% of Fortune 1000 companies (Akter & Wamba, 2016; Kiron, Prentice, & Ferguson, 2014). Nowadays, e-commerce sites in succession to adopt recommendation systems for improving customers' shopping experience (Schafer, Konstan, & Riedl, 2001). Since it is easy for customers to find plenty of low-cost competitors, it leads to the result that customers have low royalty (Sivapalan, Sadeghian, Rahnama, & Madni, 2014). To be more successful, building a personalized recommendation system for e-commerce is crucial.

In this kind of recommendation system, a significant number of visitors are likely to be newcomers with no historical data, this is known as a cold-start situation (Li, Chu, Langford, & Schapire, 2010). This makes the recommendation system difficult to learn the relation between newcomers and products and to provide newcomers good user experience. To deal with the cold-start problem, the multi-armed bandit is widely applied to balance the tradeoff between exploration and exploitation in recommendation systems (Liu, Wei, Zhang, Yan, & Yang, 2018). The multi-armed bandit is by using different models or algorithms to eliminate other fewer performance choices in order to maximize their expected gain when each choice's properties are only partially known the response of the selected prosperities at the time without the information of other none-chosen properties. With time passing by, the multi-arm bandit has a better understanding of users' choices while allocating resources to users' final choices. Such recommendation support usually involves two fundamental strategies: "Exploration", where gathering more information that might lead to better decisions in the future or "Exploitation", where making the best decision given current information. This is also named as an exploitation-exploration dilemma. There are some

DOI:10.6814/NCCU202001541

classic algorithms to solve this kind of problem (Auer & Ortner, 2010). For example, Upper Confidence Bound (UCB) is an algorithm that chooses an arm adding by the estimate reward and confidence bound of each arm. UCB is an easy way to provide a good recommendation without users' historical data. With the use of user features as context, LinUCB has made a more precise prediction when the data are sparse (Li et al., 2010). LinUCB assumes that the data are in a linear relationship, so we want to extend it to solve those data in a nonlinear relationship.

The major contributions of this study are that a) exploring non-linear relation between context and recommendations by using the NN structure; b) designing a novel training process with multiple NNs to control the effectiveness of exploration and exploitation. Since there is massive data producing by customers over time might have various of relation, using contextual bandit with that base on linear relation will meet some limits that couldn't solve nonlinear context. With the use of NNs, we could capture users' interests and improve the relevance of recommendations when the context is most of the time in a nonlinear situation. By integrating the concept of a contextual bandit with NN, we design a new algorithm that it cannot only deal with nonlinear data in a recommendation system but also highly control the different ratios between explore-exploit tradeoff. This paper provides a new NN based contextual bandit that performs as well as the other conventional algorithms. When setting different hyperparameters, the explore and exploit ratio is easily managed by the proposed algorithm. And with the design in this paper, not like the conventional algorithms the result of NNCB can be use in separate situation.

Chapter 2 includes the backgrounds and the related works. The design concept of NNCB and details are shown in Chapter 3. Chapter 4 shows the evaluation of NNCB with some conventional algorithms. The last section concludes our work and discuss the future works of this study.

DOI:10.6814/NCCU202001541

CHAPTER 2 LITERATURE REVIEW

2-1 Multi-Armed Bandits (MAB)

MAB is a simple but powerful framework for algorithms that make decisions over time under uncertainty (Zhou & Brunskill, 2016). It is a more complex version of A/B testing that uses machine learning algorithms to dynamically allocate traffic to variations. It produces faster results since it has a better way to solve explore-exploit tradeoff dilemma (Zhou & Brunskill, 2016). The purpose of MAB is to maximize its total reward. To understand which algorithms performed better, one standard approach is to compare the algorithm's cumulative reward to the actual best reward that the user could get the cumulative number is named as regret (Slivkins, 2019).

A classic MAB approach is UCB, which chooses an arm in each round by summing up the average reward and adjusting the confidence radius of each arm. When the arm is chosen with a high reward, this arm has been well exploited. When the arm is with large confidence radius, it shows that this arm has not been explored much. Both reasons make this arm worth chosen. UCB's average reward represents for exploitation and confidence radius represents for exploration so this make summing them up is a natural way to deal with the exploitation-exploration dilemma (Slivkins, 2019).

2-2 Contextual Bandits

In a real-world scenario, choosing various actions in a MAB situation, may sometimes have data that can help inform decision making like the user personal information or their previous decision. This information in the environment is called the context and the bandit problem became a contextual bandit. In some website optimization, contextual bandits use incoming user context data as it can be used to help

DOI:10.6814/NCCU202001541

make better algorithmic decisions in real-time. For example, a contextual bandit can be used to select a piece of content or ad to display on a website and the click-through rate can be used for counting regret. The context could be any information that a user gave to the algorithms, such as previously visited pages, past purchase information, device information, or geolocation. If you have news customers, and you know that a previous person with similar context coming to the site has read entertainment articles, you can select your top entertainment article to display at the top of the page for them (Chu, Li, Reyzin, & Schapire, 2011).

LinUCB extends the UCB algorithm to contextual cases. This algorithm is to compute the expected reward of each arm by finding a linear combination of the previous rewards of the arm (Li et al., 2010). Each arm is associated with a feature vector. For example, in news recommendation, articles read by users could be the feature vectors. LinUCB assumes that the expected reward of an arm is linear with respect to its feature vector and applies ridge regression to get a coefficient.

Table 2.1 Comparison between UCB, LinUCB, NNCB

Name	Feature	Contextual/Type	Regret
UCB	Use upper confidence bound to control explore-exploit tradeoff	No/NA	High
LinUCB	Use ridge regression to control explore-exploit tradeoff	Yes/Linear	Medium
NNCB	Use NN to control explore-exploit tradeoff	Yes/Linear & non-linear	Low

DOI:10.6814/NCCU202001541

CHAPTER 3　The Proposed Framework

We design a new algorithm of contextual bandit, basically by replacing the concept of matrix computing in LinUCB to Neural Network. With neural network, we can now deal with not only linear data but also non-linear data. Also, in our research that we find out NNCB in most cases can has a lower regret.

3-1 Design

In a contextual bandit problem, the most important task is to choose which arm (i.e., option) is the optimal under the real-time context (i.e., the features of a particular user). We plan to utilize multiple NNs to predict the optimal option.

The basic structure of NNCB (Neural Network Contextual Bandit, shown in Figure 1) is to dedicated NN for each option while a user context is served as the input vector of each NN which are attached after an autoencoder (AE), an artificial neural network used to learn a representation for a set of data (Goodfellow et al. 2016). The AE then reduces the dimensions of the input vector, and we adopt the latent vector inside the AE as the input of the dedicated NN for each option. The idea of using multiple NNs is that we are not able to obtain all the training labels for an input training data in the contextual bandit problem; rather, we can only obtain the label for the chosen option. A user can accept multiple options, but he or she only responds to one option at a time while training. Therefore, it is not appropriate to model a contextual bandit problem by using a multi-label classifier (because such a classifier needs all labels for input data).Another fundamental design of NNCB is a mechanism that can help to compute the confidence interval and the estimate reward for each option. The confidence interval can help us to explore more suboptimal option and gather more information from them to optimize these options. In our proposed framework we have

DOI:10.6814/NCCU202001541

two different design that can help us compute the confidence interval. One is by deployed on Bayes by Backprop mentioned in weight uncertainty in neural networks (Blundell, Cornebise, Kavukcuoglu, & Wierstra, 2015), that each time when we forward through the NN the output will slightly change. Second is by using variational autoencoders (Kingma & Welling, 2013) instead of the regular AE to reduces the dimensions of the input vector. Each time when an input vector forward through VAE it will be slightly different that could help us explore some other suboptimal options.

3-2 NNCB using Bayes by Backprop (NNCB-BNN)

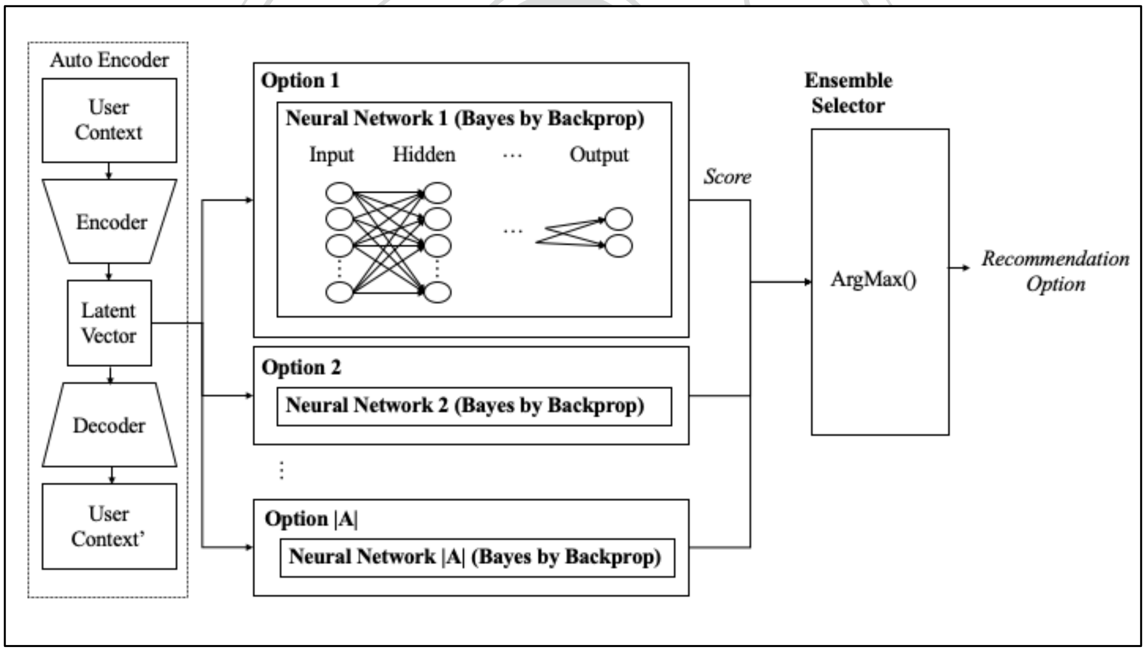

Figure 3-1 The structure of NNCB-BNN

The input will go through an AE at the beginning. Then we use the latent vaector as the input of the dedicated NN's input. With the use of Bayes by Backprop in the dedicated network we will have several estimated rewards after several runs for each option. Then we add the average and the standard deviation caculted from the output in the previous step as the final score. Among all options, the option with the highest score is chosen for this context (shown as Ensemble Selector in Figure 3-1).

DOI:10.6814/NCCU202001541

The detail of NNCB using bayes by backprop is written in Alogorithm 1. At the beginning of the training phase, all the initial weights in each NN are randomized, i.e., the output of NN is a random value; thus, the recommendation system tends to explore. In each round, the system observes a customer, and its context information is fed into the NNs to predict an optimal option for this customer. After providing a recommendation to the customer, we then obtain the feedback from the customer to check if the recommendation is a hit (1.0) or not (0.0). Since the customer only provides the response to the recommendation option, NNCB only trains the corresponding NN by using the customer's context with its response (1.0 or 0.0). Therefore, the more we choose the corresponding option we will have the more precise response we will get in other words we can exploit this option. Such design embeds the concept of exploration

Algorithm 1 NNCB using Bayes by Backprop

```
0:  Inputs: user context $U$; recommendation options $A$; the user response $r_{t,u,a}$
    Outputs: a trained model $N_a$ for each option $a$
1:  for all $N_a$ do
2:    Initialize $W_a$
3:  end for
4:  for $t = 1, 2, 3, \ldots, T$ do
5:     Observe a user context $u_t$ at $t$ using AE
6:     $MaxS \leftarrow 0$ //current maximal score
7:     $IndA \leftarrow$ null //the option index of current $MaxS$
8:     for i =1, 2, 3, …, $|A|$ do
9:         for j=1,2, 3, …, $M$ do
10:            $a_i \leftarrow A_i$
11:            $S_{a,u,j} = N(u_t)$
12:        end for
13:        $MeanS_{a,u} \leftarrow Mean(S_{a,u,1}, S_{a,u,2}, S_{a,u,3}, \ldots, S_{a,u,M})$
14:        $StdS_{a,u} \leftarrow Std(S_{a,u,1}, S_{a,u,2}, S_{a,u,3}, \ldots, S_{a,u,M})$
15:        $EstS_{a,u} \leftarrow MeanS_{a,u} + StdS_{a,u}$
16:        if ($EstS_{a,u} > MaxS$) then
17:            $MaxS \leftarrow S_{a,u}$
18:            $IndA \leftarrow i$
19:        end if
20:    end for
21:    Option $a^* = A_{IndA}$ if $IndA$ is not null else a* = rand(|$A$|)
22:    Observe a real-world binary response $r_{t,u,a}$
23:    // Activator
24:    Update the AE
25:    for i =1, 2, 3, …, $|A|$ do
26:        $a \leftarrow A_i$
27:        if $a = a^*$ then
28:            Update $N_a$ with $u_a$ as input
               and ($r_{t,u,a}$, 1.0 - $r_{t,u,a}$) as output
29:        else
30:            Update $N_a$ with $u_a$ as input
               and (0.5, 0.5) as output
31:        end if
32:    end for
33: end for
34: Output N
```

and exploitation in the NN training process which is quite important for a contextual bandit problem. It helps the NN to select exploited options and explore unchosen options at the same time.

DOI:10.6814/NCCU202001541

Assume a recommendation problem has $|A|$ options ($a \in A$) and NNCB deploys a neural network, N_a, for an option a. The trainable parameters of the neural network, N_a, is W_a. A user context is denoted as u ($u \in U$), and the dimension of user context is $|u|$. A neural network, N_a, accepts a user context as input and output score $S_{a,u}$, i.e., $N_a(u) = S_{a,u}$. $S_{a,u}$ is the confidence level (a.k.a. score) of recommending a for u. Due to the use of Bayes by Backprop the $S_{a,u}$ will be slightly different each time. Let M be the total time we want to sample the result from $N(a,u)$. Let T be the number of discrete trials of the recommendation process. The real-world response for an option a under context u is $r_{t,u,a}$ ($t \in T$) and $r_{t,u,a}$ is a binary indicator whether u selects option a (i.e., 1) or not (i.e., 0). The training process of NNCB is shown in Algorithm 1.

The inputs of NNCB algorithm are the recommendation options, the collected user contexts and their responses. At the beginning of the training phase, the parameter W_a of each NN is randomized (line 1-3). With randomized parameters, the output score of NN could be any value, which makes it possible to explore different options at the beginning phase. After the initialization, for each iteration (t), we first put the user context into the AE and we define two variables that *MaxS* stores the current highest recommendation score and *IndA* stores the index of option with highest score. We uses the latent vector inside the AE as the input of the NN in the each option. We then iteratively calculate the recommendation scores of each option with the latent vector produce from current user context (u_t) that hausing the dedicated neural networks (N_a) (line 11). And each dedicate NN will have M trial. After M trial we then calculate the corresponding $MeanS_{a,u}$ and $StdS_{a,u}$, after that we added these two number to get $EstS_{a,u}$.If the confidence level of recommending each option for the user $EstS_{a,u}$ is larger than *MaxS* (line 16), then we will update *MaxS* and *IndA*. After iterating all the options, if *MaxS* has never been updated (i.e., no option is recommended), then we will choose a random arm as the recommendation option. Otherwise, NNCB will recommend the

DOI:10.6814/NCCU202001541

IndA option which has the highest score (line 21). Then, NNCB receives the response of the recommendation. With the response, we then update the AE and neural networks for each option. For the option, a, that we recommend to the user, u, at time t, we receive its response, $r_{t,u,a}$, (either a hit (1.0) or not (0.0)) that can be directly used for updating the neural network (line 27-28). For the rest of the options, NNCB use a value of 0.5 as both recommendation and not recommendation score to update its corresponding NN (line 29-30).

3-3 NNCB using VAE (NNCB-VAE)

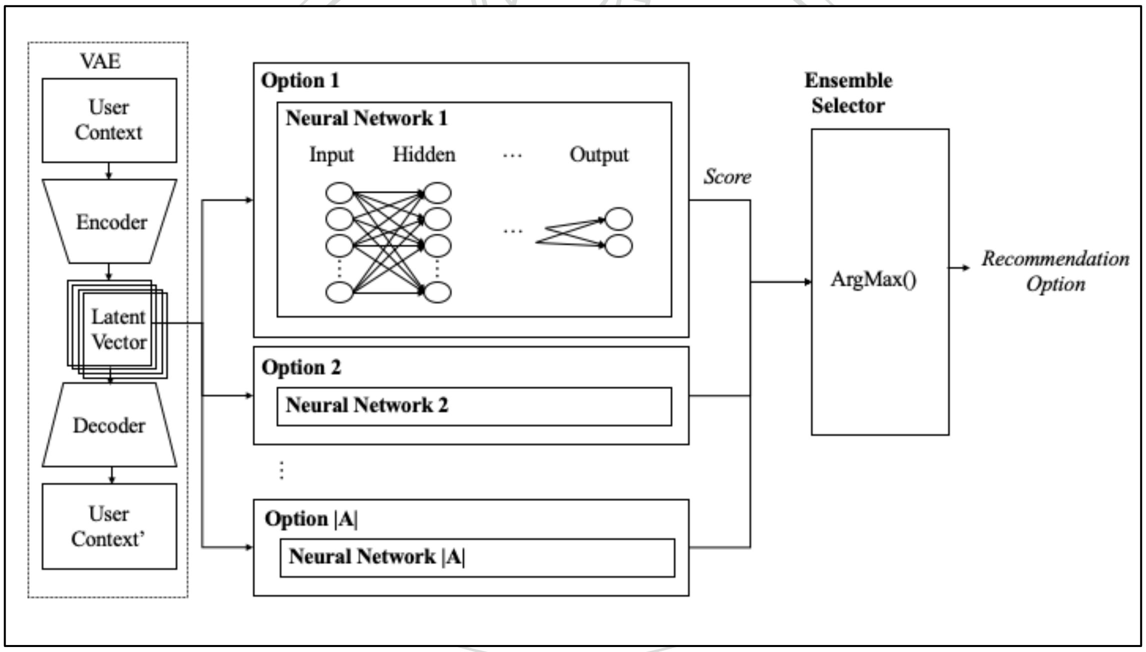

Figure 3-2 The structure of NNCB-VAE

Instead of using Bayes by Backprop we use the regular NN in this design and replace the AE to VAE. The input will go through an VAE at the beginning. Then we use the latent vaector as the input of the dedicated NN's input. For the NNCB using VAE we no longer use Bayes by Backprop to explore the suboptimal options. Fortunately, there is an efficiency way to generate the uncertainty result to help us caclute the confidence inteval when using VAE, that is when each time we passed the data through the VAE, we sample multiple number from a normal distribution that

DOI:10.6814/NCCU202001541

could help us generate different latent vector even with the same input. All of these different latent vector will be then passed to the dedicted NN for each option to calcute the the average and the standard deviation and added as the final score. Among all options, the option with the highest score is chosen for this context (shown as Ensemble Selector in Figure 3-2).

Algorithm 2 NNCB using VAE

```
0:  Inputs: user context $U$; recommendation options $A$; the user response $r_{t,u,a}$
    Outputs: a trained model $N_a$ for each option $a$
1:  for all $N_a$ do
2:      Initialize $W_a$
3:  end for
4:  for $t$ = 1, 2, 3, …, $T$ do
5:      $MaxS \leftarrow 0$ //current maximal score
6:      $IndA \leftarrow$ null //the option index of current $MaxS$
7:      for j=1,2, 3, …, $M$ do
8:          Observe a user context $u_t$ at $t$ using VAE
9:          for i =1, 2, 3, …, $|A|$ do
10:             $a_i \leftarrow A_i$
11:             $S_{a,u,j} = N(u_t)$
12:         end for
13:         $MeanS_{a,u} \leftarrow Mean(S_{a,u,1}, S_{a,u,2}, S_{a,u,3}, \ldots, S_{a,u,M})$
14:         $StdS_{a,u} \leftarrow Std(S_{a,u,1}, S_{a,u,2}, S_{a,u,3}, \ldots, S_{a,u,M})$
15:         $EstS_{a,u} \leftarrow MeanS_{a,u} + StdS_{a,u}$
16:         if ($EstS_{a,u} > MaxS$) then
17:                 $MaxS \leftarrow S_{a,u}$
18:                 $IndA \leftarrow i$
19:         end if
20:     end for
21:     Option $a^* = A_{IndA}$ if $IndA$ is not null else $a^*$ = rand($|A|$)
22:     Observe a real-world binary response $r_{t,u,a}$
23:     // Activator
24:     Update the VAE
25:     for i =1, 2, 3, …, $|A|$ do
26:         $a \leftarrow A_i$
27:         if $a = a^*$ then
28:             Update $N_a$ with $u_a$ as input
                and ($r_{t,u,a}$, 1.0 - $r_{t,u,a}$) as output
29:         else
30:             Update $N_a$ with $u_a$ as input
                and (0.5, 0.5) as output
31:         end if
32:     end for
33: end for
34: Output N
```

The initial part of NNCB-VAE algorithm is same as the NNCB using bayes by backprop. After the initialization, for each iteration (t), we define two variables that *MaxS* stores the current highest recommendation score and *IndA* stores the index of option with highest score. We then pass the current user context (u_t) through the VAE for M times. We then iteratively calculate the recommendation scores of each option using the latent vectors that generate from the previous step as the input to the dedicated neural networks (N_a). And each dedicate NN will have uses have M latent vector generate from VAE. After passing M latent vector through the NN, we then calculate

DOI:10.6814/NCCU202001541

the corresponding $MeanS_{a,u}$ and $StdS_{a,u}$, after that we added these two number to get $EstS_{a,u}$.If the confidence level of recommending each option for the user $EstS_{a,u}$ is larger than *MaxS* (line 16), then we will update *MaxS* and *IndA*. After iterating all the options, if *MaxS* has never been updated (i.e., no option is recommended), then we will choose a random arm as the recommendation option. Otherwise, NNCB will recommend the *IndA* option which has the highest score (line 21). Then, NNCB receives the response of the recommendation. With the response, we then update the VAE and neural networks for each option as what we do at the algorithm 1.

3-4 Implementation Environment

The proposed system is implemented in a PC with an Intel Core i7-7700 CPU (3.6 GHz), a GeForce GTX 1070 Ti and 16 GB DDR4 2400 RAM. The proposed neural network system is written in python 3.8 with pyTorch 1.1.1. All the comparison algorithms (including UCB, LinUCB, and Exp3) used in the experiments are modified from a public available GitHub repository [1]. The hyperparameters in the NN are optimized by AX[2], an automatic experiment platform that help test all the possible hyperparameters.

[1] https://github.com/ntucllab/striatum

[2] https://ax.dev/

DOI:10.6814/NCCU202001541

CHAPTER 4 Experimental Results

4-1 Dataset

We use Netflix Prize data[3] in the evaluation. Netflix held a Netflix Prize open competition for the best algorithm to predict user ratings for films. We reorganize the data for video recommendation. First, we sort all the users by the number of watched films and select the top ten thousand active users as the target users and choose the top 210 most watched films among the chosen users. 200 of them are chosen randomly for constructing the user context; and the rest 10 of them are used as the films for latter recommendation. A user's context is constructed by using his or her historical watching list among these 200 movies; thus, a user context is a 200-dimension vector (i.e., the number of input nodes of NNCB is 200) that each dimension specifies whether the user likes the film or not. In the experiment, we assume that a film rated by a user with at least 3 stars is considered as "recommend" in the output (i.e., $S_{a,u}$ = 1.0) or "like" (1.0) in the 200-dimension inputs by this user, and a film is considered as "not recommend" (i.e., $S_{a,u}$ = 0.0) or "dislike" (0.0) when a user rated it with less than 3 stars. For a film that a user never rates, we set the value to 0.5, and such setting is reasonable for a contextual bandit problem. For those rating that are 0.5 , if any alogrithm chose these option we will skip it and chose the second high score that has a static response which is "like" or "dislike". In all the following experiments, we compare several conventional algorithms with NNCB, and show the regret of each algorithms. We create three situation from theses dataset. The first situation is random selects a random film among 10 options in each round, that simulate the most cummon situation. The second

[3] https://www.kaggle.com/netflix-inc/netflix-prize-data

DOI:10.6814/NCCU202001541

situation is named as sorted dataset that we assume that user may be intrest of certain product because of the trend, so we sort the dataset to make all user has the same interest in certain amount of time. The third situation is named as balanced dataset that we discover that some conventinonal alogrithms tends to choose the options that has the highest excepted value, so we select certain data that makes each options' excepted value the same to see how each alogrithms performs.

4-2 Comparison between Different Algorithms

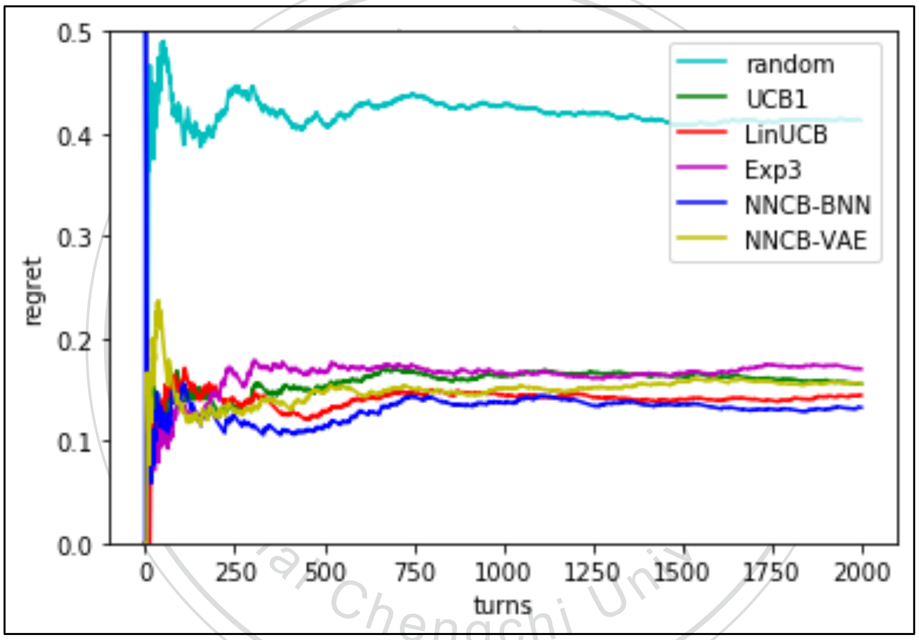

Figure 4-1 The regret of different algorithms

Figure 4-1 shows the regret of different algorithms using Netflix Prize data. NNCB-BNN has lowest regret and outperforms other algorithms. The optimal hyperparameters used in NNCB-BNN and NNCB-VAE were test by AX. All other comparison algorithms use the original version of algorithms and default settings downloaded from the GitHub repository1. For this specific dataset, to quickly identify the recommendation films for different user context is important, the NN dropout rate is set to 0.5 to avoid overfitting serveral features and helps us with faster exploration

DOI:10.6814/NCCU202001541

(hence, less exploitation). Since we have 10 options for recommendation, theoretically, a random selection algorithm will have 0.9 of regret; however, a user may have multiple preferences ("likes") among 10 options. Thus, the regret of the random algorithm for this dataset is about 0.4. It shows that on the average a user "likes" four films among these 10 options. Comparing to other algorithms, LinUCB considers the relation between the user context and the recommendation result. Like UCB1, LinUCB can quickly explore and exploit popular options at the beginning. However, when it observes more and more different user contexts, the regret increases. We anticipate that the relation between context and option may not be linear so that LinUCB needs more efforts in learning to model such non-linear relation by using a liner model. Even more, the user context in this dataset might be too complex for modeling

4-3 Dealing with Sorted Data

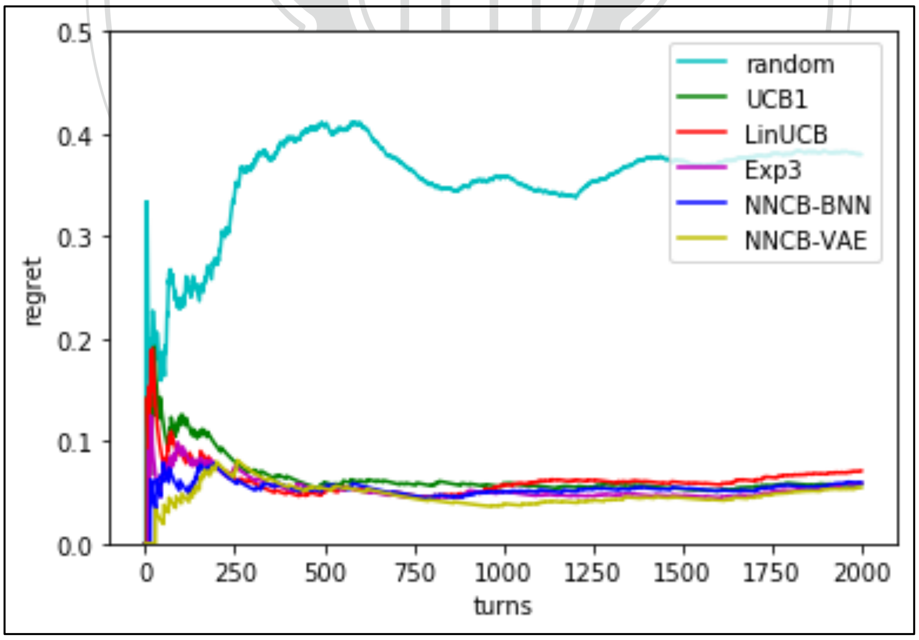

Figure 4-2 The regret of different algorithms in sorted data

As mentioned, the users may not tend to change their preference in a short period of time, thus the NNCB can quickly capture and predict the recommend option. However, in real world, film watching may have trend. To simulate trend changing, we

DOI:10.6814/NCCU202001541

sorted the users observed in each round (x axis) according to their preferences (y axis) as shown in Figure 4-3. Each dot means the user "likes" this option. The lower part of Figure 4-3 shows the recommendations made by NNCB-BNN could better fit the trend change when trend changes and NNCB-VAE tend to the choose the options that has a better result in the past. The corresponding regrets on this sorted data are shown in Figure 4-2. The blue dots in the figure indicates the successful recommendations and the red dots are the failure recommendations. The learning rate of these dataset find by AX is 0.01, we anticipate the reason of choosing big step, it can help keep recommending the same option after only a few rounds of exploration. We also note that to quickly change the recommendations, NNCB should use a simpler NN structure (e.g., a smaller number of nodes in each hidden layer or a smaller number of hidden layers) or use a simpler optimizer (e.g., SGD) (shown in Figure 3). SGD does not have momentum design which makes it quicker to train the model. This experiment shows that NNCB could deal with the trend scenario as well.

DOI:10.6814/NCCU202001541

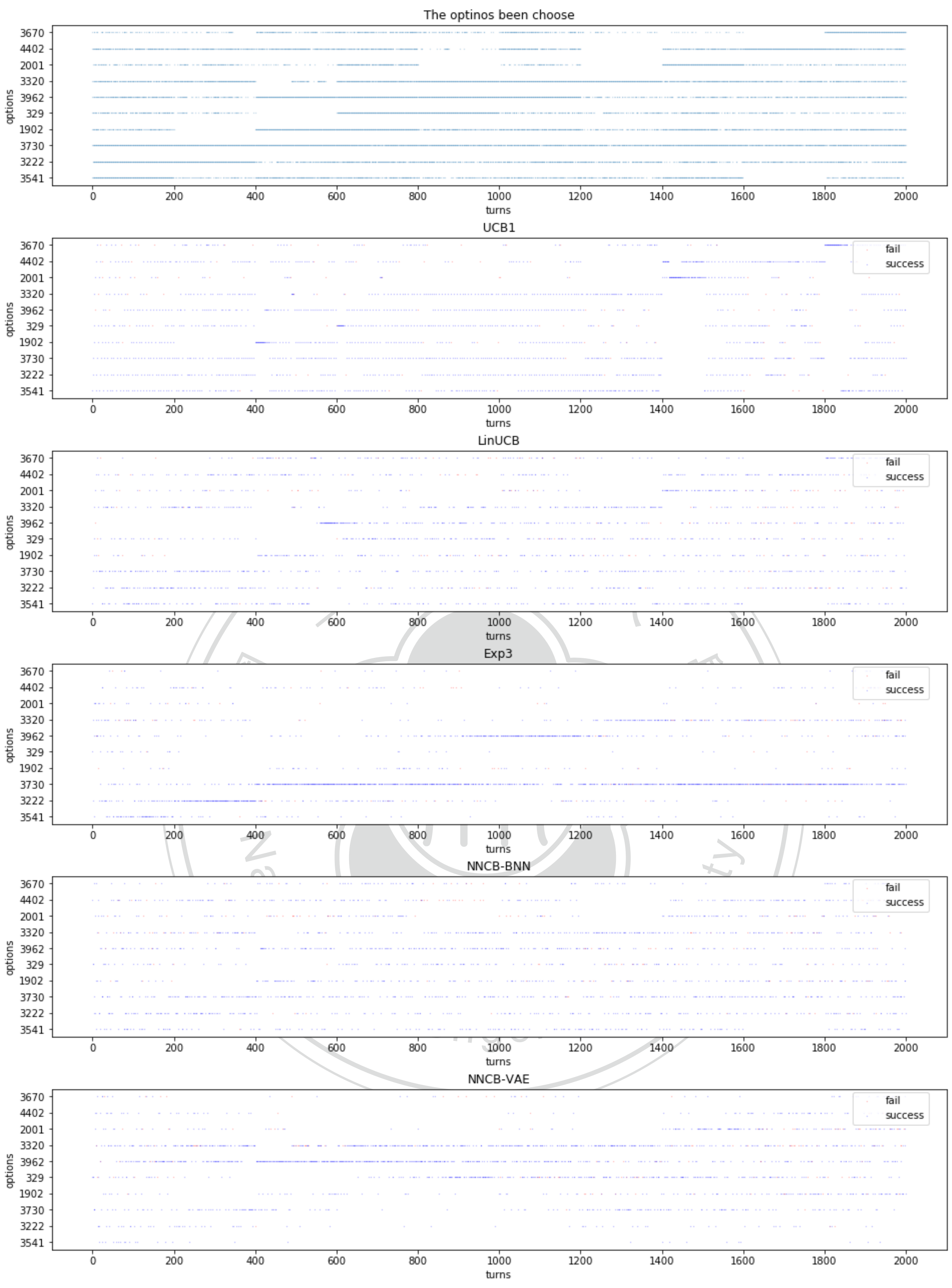

Figure 4-3 The sorted data according and the distribution of recommendations

DOI:10.6814/NCCU202001541

4-4 Dealing with Balanced Data

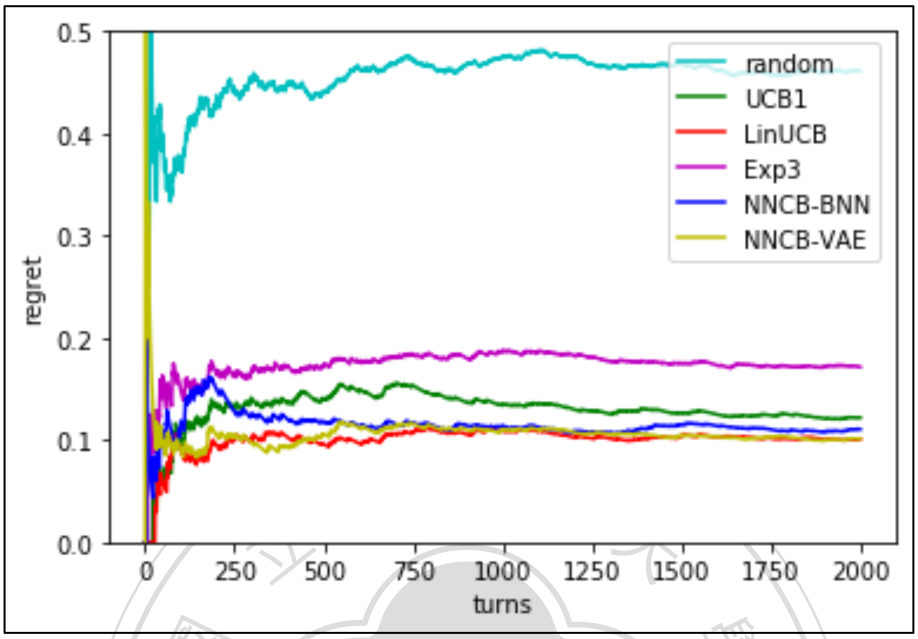

Figure 4-4 The regret of different algorithms in balanced data

Since in most experiment the LinUCB could not get the lowest regret, we then look at the options choose by each algorithm in each round. We find out that the algorithm tends to choose the options that has the most proportion and could come out with a relative low regret. So, to reduce these situations, we generate a dataset by selecting ten option out of 210 movies that has the similar proportion. These make the strategy of choosing most proportion option useless.

In figure 4-4, the LinUCB and NNCB-VAE is the one with the lowest regret. Exp3 and UCB could not find a good optimal option in the evenly distributed data due to they donesn't consider the context only choose the options with the expect values. And the NNCB-VAE could has a low regret we anticapate these is due the NN structure can learn the relation between user and the recommend movies.

DOI:10.6814/NCCU202001541

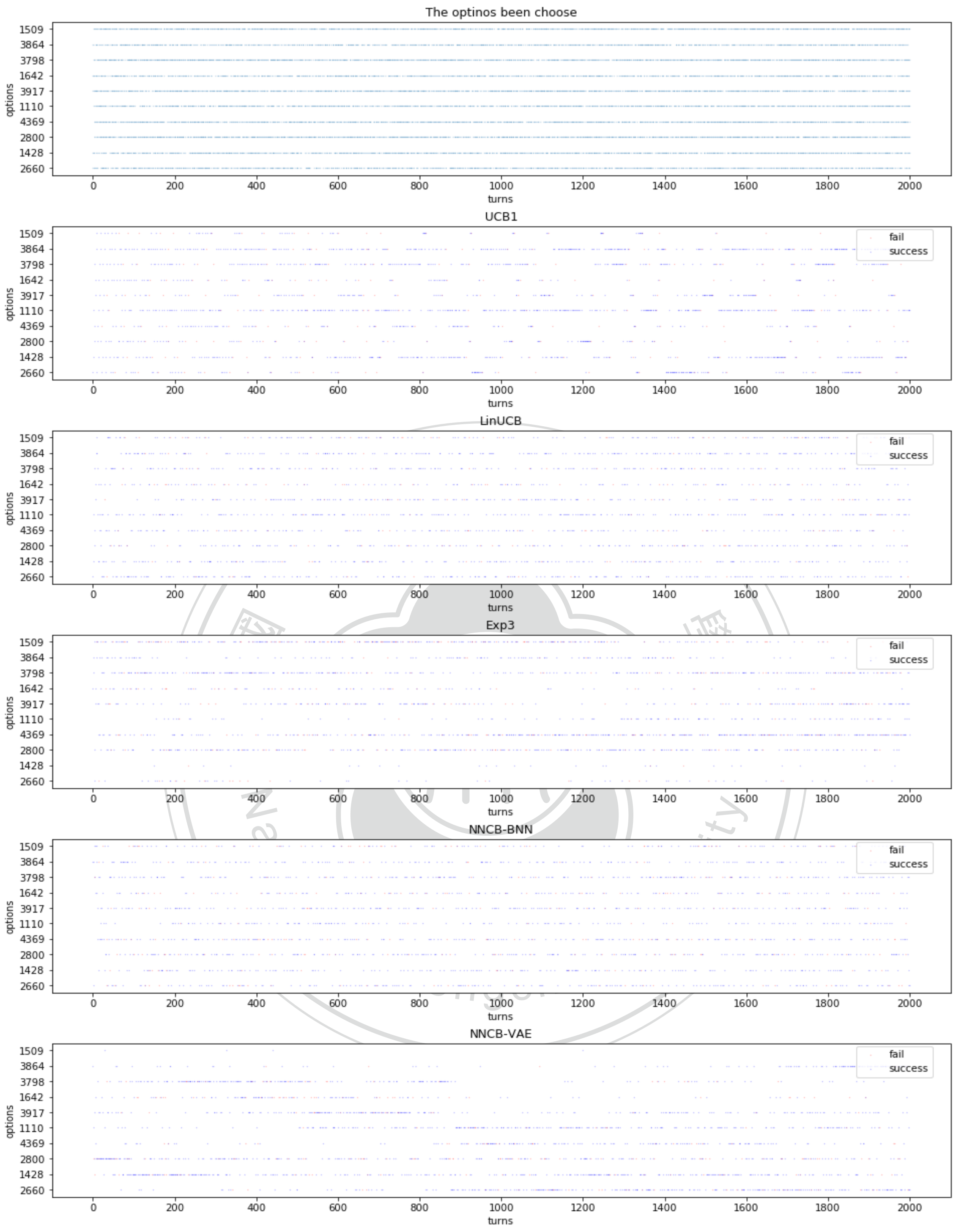

Figure 4-5 The balanced data according and the distribution of recommendations

DOI:10.6814/NCCU202001541

CHAPTER 5　Conclusion and Future Work

This paper presents a novel neural network-based contextual bandit algorithm for user recommendations. We deploy a NN for each option to generate its recommendation sore, because we anticipate that the real-world data not only in linear situation but also in non-liner, which is applicable for NN to model both situations. We leverage the initial random weights of the neural network for bandit exploration and gradually tune the NN weights by the newly observed response for exploitation. The experiment shows that the regret of NNCB outperforms the conventional contextual bandit algorithms.　In conclusion, this study shows that by applying neural networks we can better model both linear and non-linear data. By setting different dropout rates with different optimizers in the neural network, we establish a useful way to help users to set their own exploration rates. With different needs, we can now set different hyper parameters that help us best achieve the goal. And with our design of NN structure, one can easily use the needed NN that recommend options for other use. Unlike LinUCB the numbers that generate from it it can only be use for comparison inside the matrix, our NN structure generate a meanningful results, that can use to predict a user like or dislike to any movies in the options.

Although our algorithms can provide a lower or similar regret compared to other algorithms (i.e., UCB, Exp3, linUCB), there are still limitations due to the resource restraints. First, when our highest recommended option weren't selected by users, we just simply skip it and move on to the second high recommended option; thus is real world scenario these happens a lot, there's no guarantee that we have the chance to get the response of user to the options that's not at the first place. Second, the hyperparameter are all been test and found by AX. Using AX to find the best hyperparameter is time consuming, in real world scenario they may not be sufficient

DOI:10.6814/NCCU202001541

time to use AX to find the optimal hyperparameter. For future works, we will further investigate another exploration mechanism for fast changing trend, which can auto adopt to the environment. When the trend is stable that we will stop exploration, just keep exploitation, and when detected the changing of trend we then activate the model for exploration.

DOI:10.6814/NCCU202001541

REFERENCE

Akter, S., & Wamba, S. F. (2016). Big data analytics in E-commerce: a systematic review and agenda for future research. *Electronic Markets*, *26*(2), 173–194. https://doi.org/10.1007/s12525-016-0219-0

Auer, P., & Ortner, R. (2010). UCB revisited: Improved regret bounds for the stochastic multi-armed bandit problem. *Periodica Mathematica Hungarica*, *61*(1), 55–65. https://doi.org/10.1007/s10998-010-3055-6

Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015). Weight Uncertainty in Neural Networks. *Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP Volume 37. Copy- Right 2015 by the Author(S).*, *37*. https://doi.org/10.1002/etc.712

Chu, W., Li, L., Reyzin, L., & Schapire, R. E. (2011). Contextual bandits with linear Payoff functions. *Journal of Machine Learning Research*, *15*, 208–214.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *ArXiv Preprint ArXiv:1312.6114*.

Kiron, D., Prentice, P. K., & Ferguson, R. B. (2014). The Analytics Mandate. *MIT Sloan Management Review*, *55*(4), 1.

Li, L., Chu, W., Langford, J., & Schapire, R. E. (2010). A contextual-bandit approach to personalized news article recommendation. *Proceedings of the 19th International Conference on World Wide Web, WWW '10*, 661–670. https://doi.org/10.1145/1772690.1772758

Liu, B., Wei, Y., Zhang, Y., Yan, Z., & Yang, Q. (2018). Transferable Contextual Bandit for Cross-Domain Recommendation. *Aaai*, 3619–3626.

Schafer, J. Ben, Konstan, J. A., & Riedl, J. (2001). E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, *5*(1–2), 115–153.

DOI:10.6814/NCCU202001541

Sivapalan, S., Sadeghian, A., Rahnama, H., & Madni, A. M. (2014). Recommender systems in e-commerce. *World Automation Congress Proceedings*, 179–184. https://doi.org/10.1109/WAC.2014.6935763

Slivkins, A. (2019). *Introduction to Multi-Armed Bandits*. (January 2017). Retrieved from http://arxiv.org/abs/1904.07272

Zhou, L., & Brunskill, E. (2016). Latent contextual bandits and their application to personalized recommendations for new users. *IJCAI International Joint Conference on Artificial Intelligence*, *2016-Janua*, 3646–3653.

DOI:10.6814/NCCU202001541